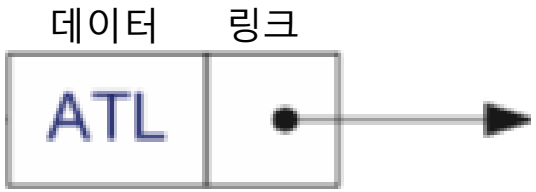
연결리스트

1 단일 링크드 리스트

1 **노드** : <**원소**,**주소**>과 같이 **원소값**과 다음 **노드 주소**를 저장하는 단위구조

데이터 필드 – 저장할 원소의 형태에 따라 하나 이상의 필드로 구성
링크 필드 – 메모리 참조 변수를 사용하여 주소에 대한 참조값 저장



<원소,주소>

2 **링크드 리스트 :** 노드들이 선형적으로 순서화된 형태의 집합체

링크드 리스트

1 단일 링크드 리스트

3 **단일 링크드 리스트:** 노드 하나가 **링크 필드에 의해서** 다음 노드와 연결되는 구조를 가진 연결 리스트

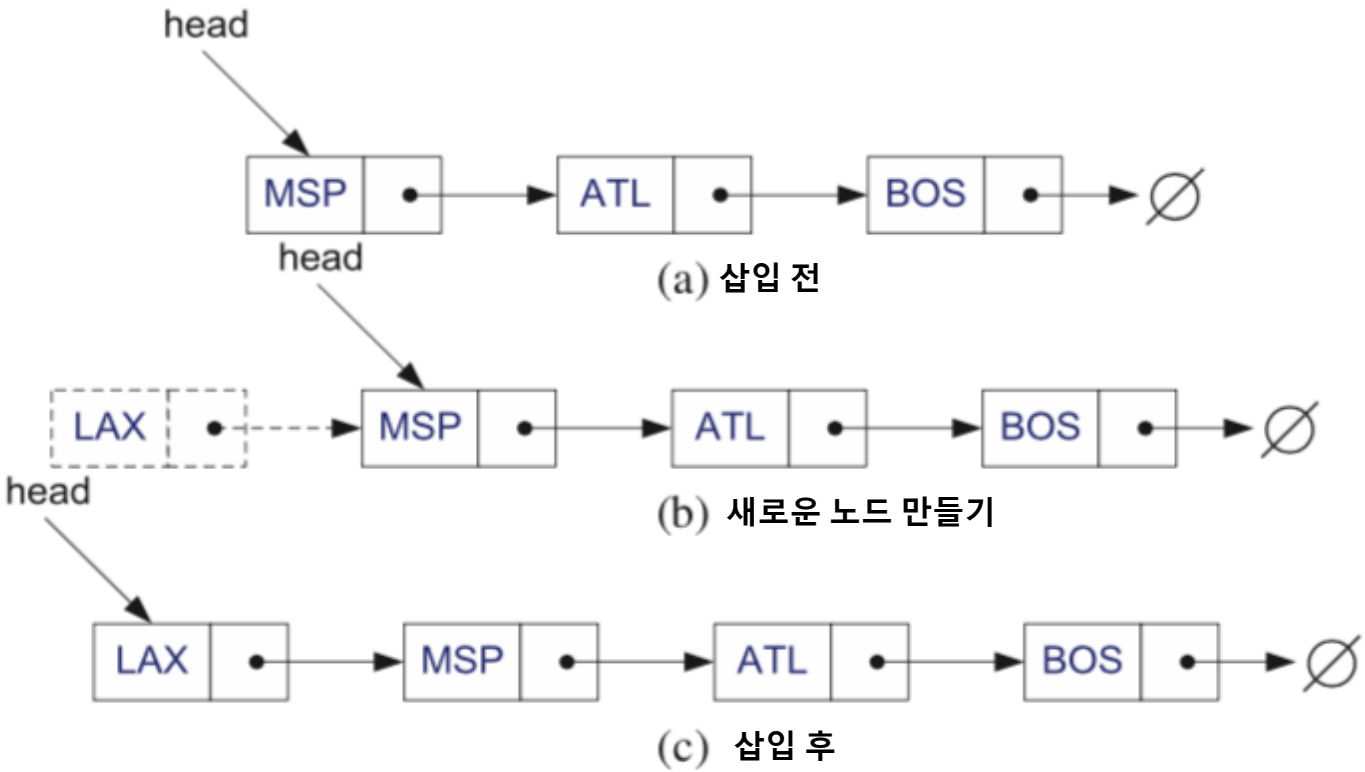
4 단일 링크드 리스트의 처음 노드와 마지막 노드를 각각 **head**, **tail**이라고 부른다.

=> null을 참조하는 next 값을 가지는 노드가 tail이다.

5 **장점** 링크드 리스트는 미리 선언되어 고정된 크기를 갖지 않는다. 노드를 추가하거나 삭제함으로써 사이즈를 조정할 수 있다.

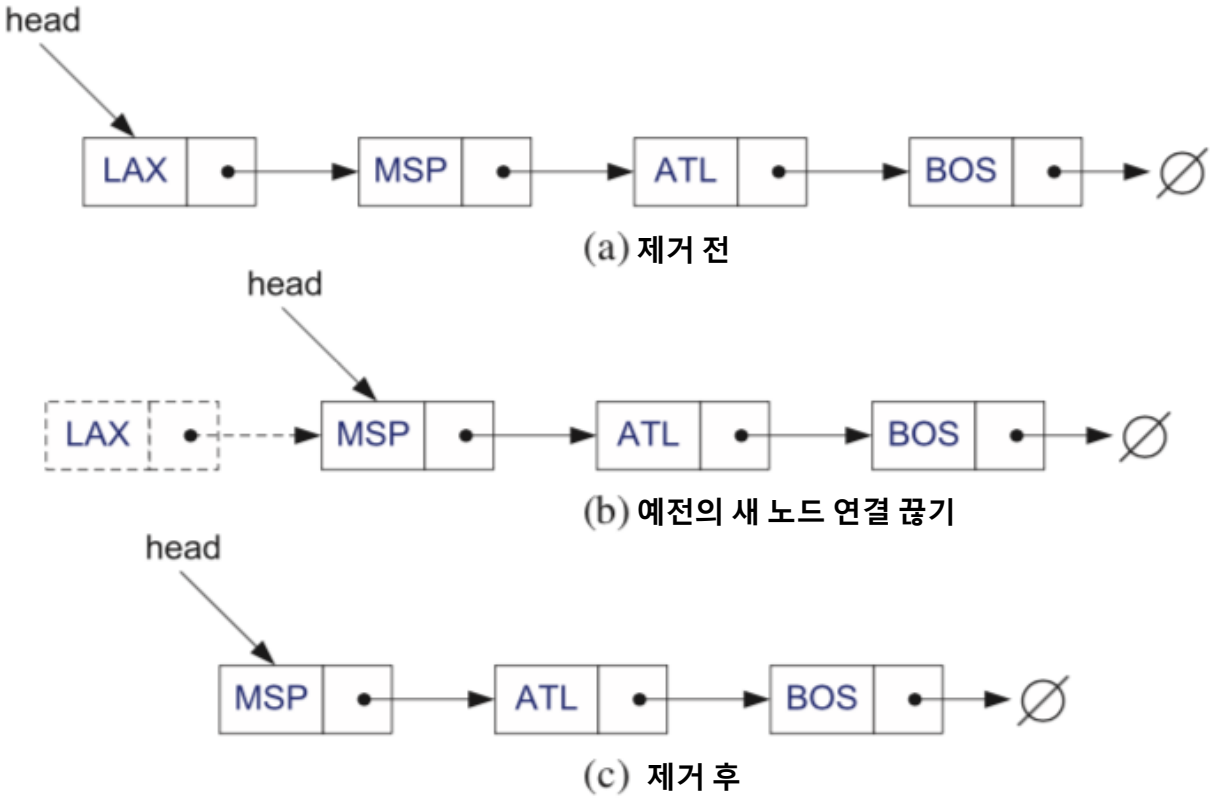
단점 index를 이용해 원하는 element에 바로 접근하는 배열과 달리 원하는 element에 접근하기 위해 순차적으로 접근 해야 한다.

6 단일 링크드 리스트의 **앞에 삽입** addFront(element)



Cost : constant - time

7 단일 링크드 리스트의 **앞에서 삭제** removeFront()



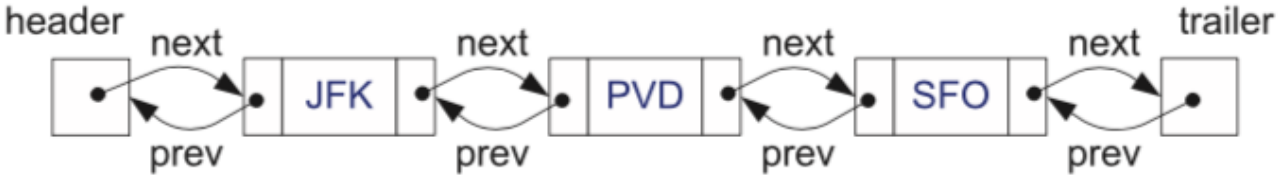
Cost : constant - time

2 이중 링크드 리스트

1. 단일 링크드 리스트의 탐색 기능을 개선한 자료구조

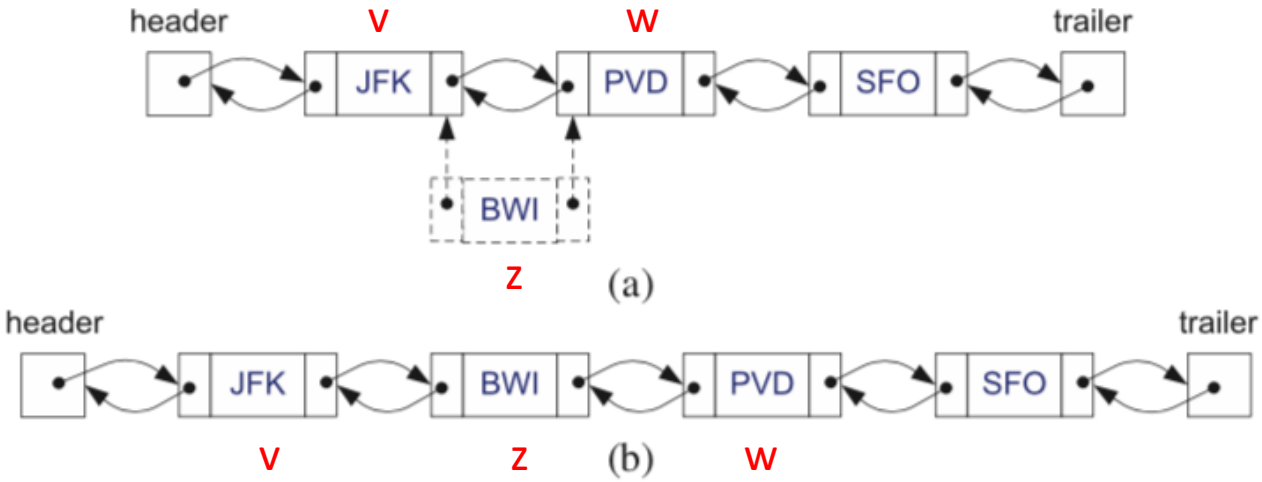
2. 단일 링크드 리스트의 노드가 다음 노드뿐 만 아니라 **이전 노드를 가리키는 포인터**를 갖고 있음.

3. 단일 링크드 리스트와는 달리 **양방향 탐색**이 가능함.



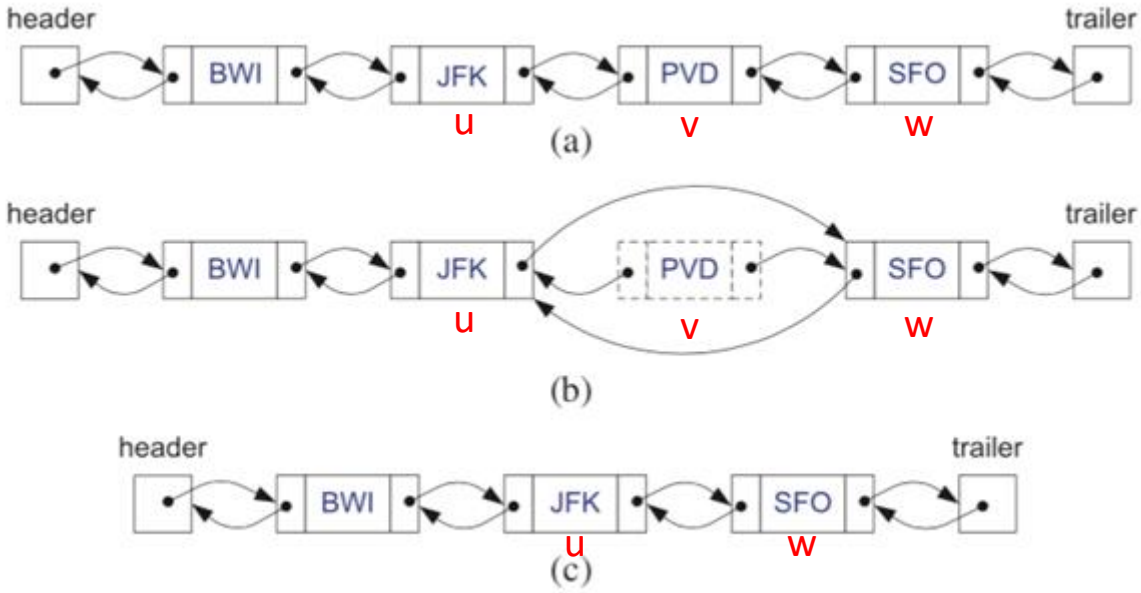
링크드 리스트

4 이중 링크드 리스트에 **삽입**



- z의 prev 링크는 v를 가리키도록 한다.
- z의 next 링크는 w를 가리키도록 한다.
- w의 prev 링크는 z를 가리키도록 한다.
- v의 next 링크는 z를 가리키도록 한다.

5 이중 링크드 리스트에서의 **삭제**



- w의 prev 링크는 u를 가리키도록 만든다.
- u의 next 링크가 w를 가리키도록 만든다.
- v 노드를 삭제한다.

3 환형 링크드 리스트

1. 단일 링크드 리스트의 **tail와 head**가 next 포인터를 이용하여 연결되어 있는 형태

2. **Cursor**라고 불리우는 노드를 갖고 있어서, 환형 링크드 리스트를 탐색할 때에 시작할 지점을 정해준다.

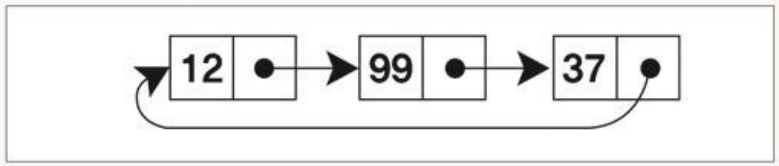


그림 1. 선형과 원형 리스트(linear and circular list)

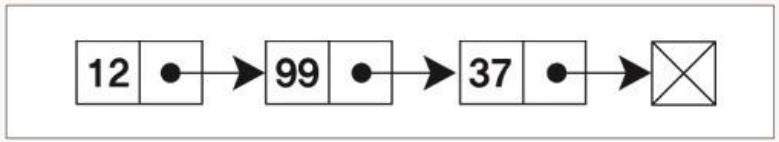


그림 2. Singly linked list

링크드 리스트

```
class Node {
public:
    int data;     // 원소값
    Node* next;   // 다음 노드 주소

    Node(int e) {
        this->data = e;
        this->next = NULL;
    }
};

class SLinkedList {
public:
    Node* head;   // 리스트의 head
    Node* tail;   // 리스트의 tail

    SLinkedList() {
        head = NULL;
        tail = NULL;
    }
```

링크드 리스트

```
void addFront(int X) {   // 리스트의 가장 앞에 노드 삽입
                         // new Node 할당
    if                 { // 리스트가 비어 있을 때

    }
    else {


    }
}
int removeFront() {     // 리스트의 가장 앞 노드 삭제 후 원소값 반환
    if                  // 리스트가 비어 있을 때
    else {


    }
}
int front() {   // 리스트의 가장 앞 노드의 원소값 반환
    if          //리스트가 비어 있을 때
    else
}
int empty() {   // 리스트가 비었는지 확인하는 값 반환
    if
    else
}
```

링크드 리스트

```
void showList() {  // 리스트에 저장되어 있는 정수들을 앞에서부터 차례로 출력
    if                                    // 리스트가 비어 있을 때
    else {
    }
}
void addBack(int X) {  // 리스트의 가장 뒤에 노드 삽입
                                  // new Node 할당
    if                  {  // 리스트가 비어 있을 때
    }
    else {
    }
}
```


Thank you